```cpp
class Parent
{
public:
    void M1()
    {
        cout << "Parent::M1" << endl;
    }
    virtual void M2()
    {
        cout << "Parent::M2" << endl;
    }
    void M3()
    {
        cout << "Parent::M3" << endl;
        M4();
    }
    virtual void M4()
    {
        cout << "Parent::M4" << endl;
    }
};

class Child : public Parent
{
public:
    void M1()
    {
        cout << "Child::M1" << endl;
    }
    virtual void M2()
    {
        cout << "Child::M2" << endl;
    }
    virtual void M4()
    {
        cout << "Child::M4" << endl;
    }
};

void Caller1( Parent *p )
{
    p->M1();
    p->M2();
}

void Caller2( Parent *p )
{
    p->M3();
}

void TestVirtual()
{
    Parent p;
    Caller1( &p ); // Parent::M1\nParent::M2
    Child c;
    Caller1( &c ); // Parent::M1\nChild::M2

    Caller2( &p );
    Caller2( &c );
}

typedef Parent * PParent;
void TestMemory()
{
    {
        int *p = new int;
        *p = 100;
        cout << *p << endl;
        delete p;
        p = 0;

        Parent *base = new Child();
        base->M1();
        base->M2();
        delete base;
        base = 0;
    }
    {
        int *p = new int[10];
        for( int i = 0; i < 10; ++i )
        {
            p[i] = i * i;
        }
        delete[] p;
        p = 0;

        // Create using default constructor
        Parent *bases = new Parent[5];
        for( int i = 0; i < 5; ++i )
        {
            bases[i].M1();
            bases[i].M2();
        }
        delete[] bases;
        bases = 0;

        // 5 pointers to Parent, no memory allocation
        // Parent *bases2[5];
        PParent bases2[5];
        cout << "Virtual test" << endl;
        for( int i = 0; i < 5; ++i )
        {
            if ( ( i % 2 ) == 0 )
            {
                bases2[i] = new Parent();
            }
            else
            {
                bases2[i] = new Child();
            }
            bases2[i]->M1();
            bases2[i]->M2();
        }

        for( int i = 0; i < 5; ++i )
        {
            delete bases2[i];
            bases2[i] = 0;
        }
    }
}

class ClassA
{
public:
    ClassA()
    {
        cout << "ClassA::ClassA" << endl;
    }
    virtual ~ClassA()
    {
        cout << "ClassA::~ClassA" << endl;
    }

};

class ClassB : public ClassA
{
public:
    int *p;
    ClassB() :
        ClassA()
    {
        p = new int[10];
        cout << "ClassB::ClassB, new 10 ints" << endl;
    }
    virtual ~ClassB()
    {
        delete[] p;
        cout << "ClassB::~ClassB, free 10 ints" << endl;
    }

};

void TestDestructors()
{
    {
        ClassA a;
        ClassB b;
    }
    {
        ClassA *base = new ClassB();
        // If destructor is not virtual then
        // ~ClassA will be called  and memory is not freed
        delete base;
    }
}

void ChangeIntPointer( int *x )
{
    *x = 100;
}

void ChangeIntRef( int &x )
{
    x = 200;
}

void Caller3( Parent &p )
{
    p.M1();
    p.M2();
}

void TestReferences()
{
    int x;
    ChangeIntPointer( &x );
    cout << x << endl;
    ChangeIntRef( x );
    cout << x << endl;

    int *pX = &x;
    *pX = 300;
    cout << x << " " << *pX << endl;
    int &refX = x;
    refX = 400;
    cout << x << " " << refX << endl;

    Parent p;
    Child c;
    Caller3( p );
    Caller3( c );
    Parent *p2 = new Child();
    Caller3( *p2 );
    delete p2;
    p2 = 0;
    //Caller3( *p2 ); // runtime error
}
```

```cpp
const int FPS = 60;
const int SCREEN_W = 640;
const int SCREEN_H = 480;

class Figure
{
protected:
    double x_;
    double y_;
    double dx_;
    double dy_;

public:
    Figure()
    {
        Reset();
    }

    void Reset()
    {
        x_ = rand() % SCREEN_W;
        y_ = rand() % SCREEN_H;
        dx_ = 10.0 - rand() % 21;
        dy_ = 10.0 - rand() % 21;
    }

    virtual void Draw(){}

    virtual void Move()
    {
        x_ += dx_;
        y_ += dy_;
        if ( ( x_ < 1.0 ) || ( x_ > SCREEN_W ) ||
             ( y_ < 1.0 ) || ( y_ > SCREEN_H ) )
        {
            Reset();
        }
    };
};
typedef Figure * PFigure;

class Square : public Figure
{
protected:
    double a_;
public:
    Square( double a ) :
        Figure(),
        a_( a )
    {
    }
    virtual void Draw()
    {
        double half = a_ / 2;
        al_draw_filled_rectangle( x_ - half, y_ - half,
            x_ + half, y_ + half, al_map_rgb( 255, 0, 0 ) );
    }
};

class Circle : public Figure
{
protected:
    double r_;
    unsigned char color_;
public:
    Circle( double r ) :
        Figure(),
        r_( r ),
        color_( rand() % 256 )
    {
    }
    virtual void Draw()
    {
        ++color_;
        al_draw_filled_circle( x_, y_, r_, al_map_rgb( 0,
color_, 0 ) );
    }
};

const int MAX = 100;
class ScreenSaver
{
```

```cpp
private:
    PFigure figures[MAX];
    int size_;

public:
    ScreenSaver() :
        size_( 0 )
    {
        // Set to null all pointers
        memset( figures, 0, sizeof( figures ) );
    }
    ~ScreenSaver()
    {
        for( int i = 0; i < size_; ++i )
        {
            delete figures[i];
            figures[i] = 0;
        }
    }

    void Draw()
    {
        al_clear_to_color( al_map_rgb( 0, 0, 0 ) );
        for( int i = 0; i < size_; ++i )
        {
            figures[i]->Draw();
        }
    }
    void Next()
    {
        for( int i = 0; i < size_; ++i )
        {
            figures[i]->Move();
        }
    }

    void Add( Figure *f )
    {
        if ( size_ >= MAX )
        {
            return;
        }
        figures[ size_ ] = f;
        ++size_;
    }

};

ScreenSaver ss;

void fps()
{
    ss.Next();
}
void draw()
{
    ss.Draw();
}

int main(int argc, char **argv)
{
    srand( time(0) );
    if( !InitAllegro( SCREEN_W, SCREEN_H, FPS ) )
    {
        DestroyAllegro();
        return 1;
    }
    for( int i = 0; i < 100; ++i )
    {
        if ( ( i % 2 ) == 0 )
        {
            ss.Add( new Circle( 10.0 + rand() % 30 ) );
        }
        else
        {
            ss.Add( new Square( 10.0 + rand() % 30 ) );
        }
    }
    RunAllegro( &fps, &draw );
    DestroyAllegro();
    return 0;
}
```